INTEGRATED SOFTWARE

The development of a very large scale integrated (VLSI) circuit involves many steps such as logic design and simulation, circuit design, layout design, simulation, fabrication and testing (1). A VLSI circuit chip either to be marketed or put in some application has to be tested to locate defects and malfunctions. Testing in the context of digital systems is defined to be the process by which a defect in the system can be exposed. This is done by observing the response of a digital system to an input stimulus. If the expected response of the system is known, then the device that is being tested is determined to be defective or otherwise by comparing the actual response with the expected response. A failure detected in testing can be described as a lack of expected performance. An instance of an incorrect operation of a digital system or a component may be defined to be an error. An error observed by an automatic testing equipment (ATE) for testing digital systems implies incorrect binary value. The cause of error may be improper design (design error), imperfect manufacturing process (fabrication error), or failures due to wear-out of components (physical failure). The cause of an error is called the *fault*. A fault which can change the logic value on a line in the circuit from logic 0 to logic 1 or vice versa is called a *logic fault*. Logical faults in a digital circuit can be detected and located by the application of digital stimulus vectors and observing the response. Thus for testing a digital system implemented in a VLSI circuit, we need a set of stimulus vectors which are forced on the input pins of the VLSI circuit. The expected output response from the system is then used to compare with the actual response of the VLSI circuit being tested (2,3). If the comparison shows any discrepancies, the testing process is carried over to diagnosis stage, where the faults are located on the chip. The corresponding parts are then modified and the chip is refabricated incorporating the modifications. Figure 1 shows the hierarchy of processes of a typical VLSI circuit development which includes testing.

The design of a system or subsystem on a silicon chip begins with its functional specification as shown in Fig. 1. These specifications are converted into a layout design in a particular technology [e.g., complementary metal-oxide-semiconductor (CMOS) technology] in a top-down design approach (1) involving abstraction at different levels. In hierarchy of a typical chip design process, two intermediate levels in the design abstraction are designs at the logic and circuit levels. Digital simulations are done both at the logic and circuit levels to verify the logic design and performance. The circuit level design is converted into a layout design for the patterning process on silicon. The SPICE (Simulation Program with Integrated Circuit Emphasis) (4) netlist is extracted from the chip layout which includes parasites such as node capacitances. The modification is done to the extracted netlist to include device models and input test vectors. This netlist is simulated using SPICE simulator for the chip performance. The SPICE simulated results are compared with specifications defined at the system level. In case of any discrepancy, appropriate modifications are incorporated in logic and/or circuit designs to meet the design specifications.

The chip is fabricated using the layout design information as an input and tested for its performance using the developed test programs. The type and nature of faults are determined if the design does not meet specifications. The fault may be related either to the process technology or the design. Depending on the type and nature of the fault, the information is fed to the fabricator and/or the layout designer. The chip design is corrected, fabricated, and tested. The process stops when the chip meets its functional specifications defined at the system level.

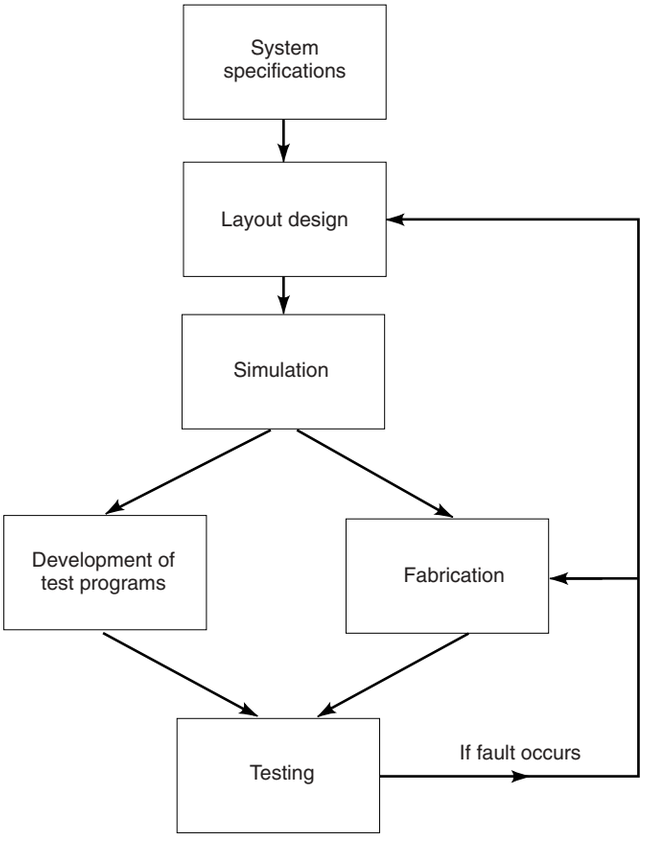

Figure 1. Hierarchy of processes in VLSI Chip development. (Reprinted with the permission of IEEE and of Gordon and Breach Publishers.)

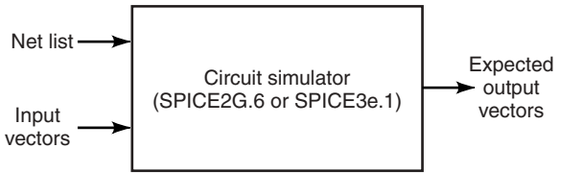

Figure 2. VLSI circuit simulation.

Figure 2 shows a typical VLSI circuit stimulation process. The input stimulus vectors and the netlist extracted from the VLSI circuit layout are given to a simulator. A simulator is a program that models the network being studied (4). Such circuit states, as faults, whose realization in real circuits is difficult, may easily be realized in simulation environment. The netlist required as the body of the program for a VLSI circuit simulator can be derived from a VLSI layout design tool. A VLSI circuit simulator derives the sequence of the output vectors that are expected to result on applying a particular sequence of input vectors on a VLSI circuit that is being simulated. Thus, the simulator simulates the circuit in hand using the circuit definition file which consists of device equivalent circuit models and other network parameters. The simulator then runs analysis for the circuit definition file and gives the output response corresponding to a defined set of dc and/or pulsed binary input parameters. These dc and/or pulsed binary input parameters and corresponding generated output response by the simulator are termed *input* and *output* vectors. The vectors needed for testing are thus generated by simulation, but these vectors are in a format specific to the simulator used. The tester may need them in a different format with additional information such as clocking information. All these needs are to be provided by the testing engineer. This process consumes time and is error-prone since human factor is involved.

The design automation and testing process of a VLSI circuit through an interface between a simulator and a design verification system enhances the efficiency of testing since the involvement of human factor is greatly reduced. The automation of testing process also reduces the time consumed since the test program for the VLSI tester is generated by the interface. The present work involves integration of the simulation stage of design of a VLSI circuit and its testing stage (5,6). The SPICE simulator, TEK LV500 ASIC Design Verification System, and TekWAVES, a test program generator for LV500, were integrated. A software interface in "C" language in UNIX "solaris1.x" environment has been developed between SPICE and the testing tools (TekWAVES and LV500). A graphical user interface has also been developed with OPENWINDOW'S using Xview toolkit. As an example, a two-phase clock generator circuit has been considered and usefulness of the software demonstrated.

SOFTWARE INTERFACE DEVELOPMENT

Figure 3 shows the data flow diagram of the present work which integrates the SPICE simulator, TEK LV500 ASIC Design Verification System, and TekWAVES. The input pulses needed are taken from input files of SPICE simulator, and output pulses are extracted from SPICE output. All formats of input vectors that result in generation of digital pulses are converted into EWAV (event wave) format. The output pulses corresponding to various output nodes specified in SPICE input file are found in SPICE output file and are converted into EWAV format.

CAD Tools Selected for Integration

The design verification system consists of "TEK LV500 ASIC Design Verification System" and "TekWAVES," respectively. The LV500 is a stand-alone design verification tester for application specific integrated circuits (7). It provides 64 to 256 bidirectional tester channels at test speeds up to 50 MHz. In this work, the tester used provides 64 channels. The LV500 takes input stimulus vectors and expected output vectors for a particular integrated circuit under test in the form of a test program. It tests the chip mounted on it by comparing the actual output vectors obtained from a fabricated integrated circuit (IC) chip and the expected output vectors fed to it, and shows the discrepancies between the two, if any. The TekWAVES is a software package by which digital stimulus vectors can be created, viewed and manipulated (8). It aids LV500 in ASIC design verification. With TekWAVES, the test programs can be acquired and downloaded to and from LV500. The TekWAVES needs the input vectors and expected output vectors in an event wave (EWAV) format as an input. The EWAV format is explained in a later part of this section.

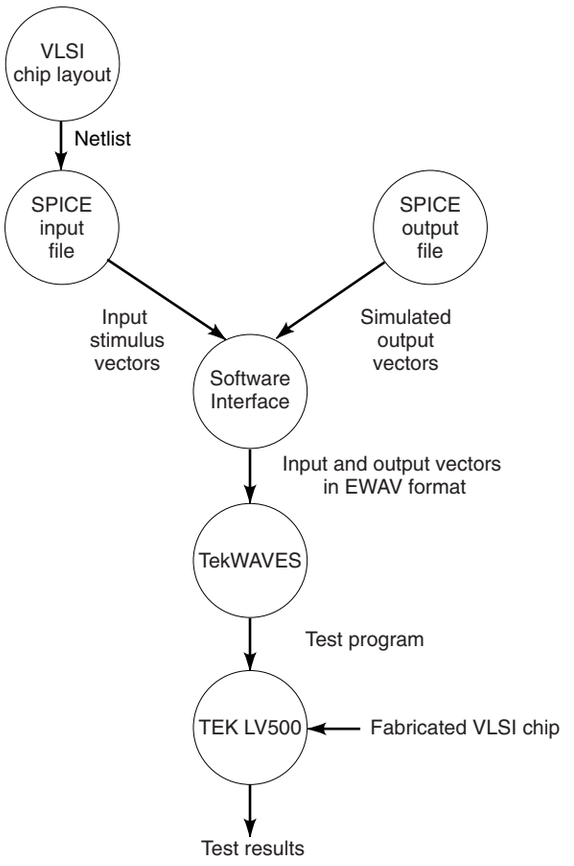

Figure 3. Data flow diagram. (Reprinted with the permission of IEEE and of Gordon and Breach Publishers.)

The simulation tool SPICE is most commonly used for circuit simulation. It takes the input vectors and the netlist extracted from an integrated circuit layout, in a code specific to it, and generates the expected output vectors that are printed into an output file, generated in the process. Two versions of SPICE (SPICE2G.6 and SPICE3e.1) are considered in the present work. SPICE3e.1 takes the input file in the same format as that of SPICE2G.6, but prints out the output in a different format. The output file of SPICE3e.1, is called rawfile. Tables 1 and 2 show the format of input–output files.

The EWAV File Format

The EWAV file format is an ASCII format for event files for logic simulators and testers. The EWAV file format has three sections. The first section is the "environment" section, the second section is the "signal declaration" section, and the final is the "event data" section. The input and output signals of the VLSI circuit in hand are declared in the signal declaration section. The event data section is the section in which the input and output vectors are specified. The format of an EWAV file is shown below:

The Environment Section

version (required)
date (optional)
time (optional)
the creator (optional)
the intended destination (optional)
timescale (optional, but 1ps is the default)

The Signal Declaration Section

signal name_of_the_signal directionality {
path (optional)
polarity (optional)
}

The Event Data Section

timestamp: event data; (for timestamped data vectors)
timestamp@ event data with signal keys specified for changed signals; (for timestamped signal change list)

The Environment Section. This section contains information such as an EWAV file version, the data created, the time created, the creator, the intended destination, and finally the timescale. Any other information than version and timescale is optional. The timescale statement defines the units of the ones digit for the event time found in each vector in the event data section of the EWAV file. An example of the timescale statement is shown in the example EWAV file in Table 3. A default timescale of 1 ps is assumed if timescale is not specified.

The Signal Declaration Section. The signal declaration section is a list of signal definitions. The order of the signal declarations is significant. The event data are grouped in the same order as the signal definitions.

A signal definition has a signal name, an optional bit specification, a direction, an optimal path, and finally an optional bi-directional reference. Refer to the EWAV file format in the EWAV file format section.

The legal directions possible are input, output, and bidir. Data for a signal of the input direction are forced into the device under test (DUT). Data for a signal of the output direction are compared with the results from the DUT. Data for a signal of the bidir direction are both force and compare.

The path statement is inside the curly braces of a signal statement. It enables us to include the hierarchical path name as part of name of the signal. An example path statement is

path = "/cpu/cell12"

The reference statement is inside the curly braces of a bidirectional signal statement. It enables us to specify the bidirectional reference for the signal. This determines the direction for the signal specified by using the data in the reference signal. An example of the reference statement is

reference read {polarity=positive}

The Event Data Section. The event data section has three types of statements. They are timestamped data vector, timestamped signal change list, and markers.

Each timestamped vector contains the time (in timescale units) when the change occurred. The time is followed by a colon. Following the colon is a list of binary values for the signals, which are listed in the order of the signals in the signal section. The vector is ended with a semicolon. An example timestamped data vector is

100: 1011 LHLLHLH;

The timestamped signal change list notates small changes in data vectors in more compact fashion. It allows only signals with changes to be listed. It has a timestamp, in timescale units, followed by an at-sign (@).

Each signal data change in the signal change list is notated with a signal key followed by a comma (,) and then the data for the signal. A signal key is the order number that a signal appears in the signal declaration section (8). The key of the first signal would be 0, the key of the next signal would be 1, and so on. There may be any number of key-data pairs on the line of the signal change list. A semicolon (;) follows the last key-data pair. An example of a signal change list is

1020@ 1,1011 4, HLHHHLH;

This example says that the second and fifth signals have changed at time 1020.

A marker in the event data section is a comment. It is a way to notate some interesting thing about the data in a way that may be kept after the EWAV file has been transferred into another file format. C-style comments which are embedded within /* and*/ are generally removed by most translators, while markers are passed through the translation process. An example of a marker statement is

marker "read cycle starts"

Table 1. SPICE2G.6 Output File

```
0Clock Generator
0****                    Input Listing                    Temperature = 27.000 DEG C
0******************************************************************************
M1    3   2   1   1    MOD1   W = 3U   L = 3U
M2    6   3   1   1    MOD1   W = 3U   L = 3U
M3    5   4   6   1    MOD1   W = 3U   L = 3U
M4    7   5   1   1    MOD1   W = 3U   L = 3U
M5    4   2   7   1    MOD1   W = 3U   L = 3U
M6    3   2   0   0    MOD2   W = 3U   L = 3U
M7    5   3   0   0    MOD2   W = 3U   L = 3U
M8    5   4   0   0    MOD2   W = 3U   L = 3U
M9    4   5   0   0    MOD2   W = 3U   L = 3U
M10   4   2   0   0    MOD2   W = 3U   L = 3U

VDD   1   0   DC        5
VIN1  2   0   PULSE (0   5   0NS   2NS   2NS   100NS   200NS)

.MODEL   MOD1   PMOS
.MODEL   MOD2   NMOS

.TRAN   20NS   1080NS
.PRINT   TRAN   V(3)   V(5)   V(7)
.END

1**************06/11/97**************SPICE2G.6***************************
0Clock Generator
0***                     Transient Analysis              Temperature = 27.000 DEG C
0******************************************************************************
     Time                V(3)                V(5)                V(7)

0.000E + 00          5.000E + 00         3.467E − 08         5.000E + 00
2.000E − 08          6.933E − 08         5.000E + 00         5.000E + 00
4.000E − 08          6.933E − 08         5.000E + 00         5.000E + 00
6.000E − 08          6.933E − 08         5.000E + 00         5.000E + 00
8.000E − 08          6.933E − 08         5.000E + 00         5.000E + 00
1.000E − 07          6.933E − 08         5.000E + 00         5.000E + 00
1.200E − 07          5.000E + 00         3.464E − 08         5.000E + 00
1.400E − 07          5.000E + 00         3.465E − 08         5.000E + 00
1.600E − 07          5.000E + 00         3.467E − 08         5.000E + 00
1.800E − 07          5.000E + 00         3.467E − 08         5.000E + 00
2.000E − 07          5.000E + 00         3.467E − 08         5.000E + 00
2.200E − 07          6.933E − 08         5.000E + 00         5.000E + 00
2.400E − 07          6.933E − 08         5.000E + 00         5.000E + 00
2.600E − 07          6.933E − 08         5.000E + 00         5.000E + 00
2.800E − 07          6.933E − 08         5.000E + 00         5.000E + 00
3.000E − 07          6.933E − 08         5.000E + 00         5.000E + 00
3.200E − 07          5.000E + 00         3.464E − 08         5.000E + 00
3.400E − 07          5.000E + 00         3.465E − 08         5.000E + 00
3.600E − 07          5.000E + 00         3.467E − 08         5.000E + 00
3.800E − 07          5.000E + 00         3.467E − 08         5.000E + 00
4.000E − 07          5.000E + 00         3.467E − 08         5.000E + 00
4.200E − 07          6.933E − 08         5.000E + 00         5.000E + 00
4.400E − 07          6.933E − 08         5.000E + 00         5.000E + 00
4.600E − 07          6.933E − 08         5.000E + 00         5.000E + 00
4.800E − 07          6.933E − 08         5.000E + 00         5.000E + 00
5.000E − 07          6.933E − 08         5.000E − 00         5.000E + 00
5.200E − 07          5.000E + 00         3.464E − 08         5.000E + 00
5.400E − 07          5.000E + 00         3.465E − 08         5.000E + 00

                              JOBCONCLUDED
0******************************************************************************
```

Pin Directionality in EWAV. The data for signals with direction inputs is either a 1 for high, a 0 for low, a "z" for high impedance, or an "x" for unknown. The data for signals with the direction output is either an "H" for high, an "L" for low, a "T" for high impedance, or an "X" for unknown. The data for bi-directional signal is specified in input signal format or output signal format depending on whether the signal is used for force or for compare.

Table 2. SPICE3e.1 Output File

Title: *CLOCK GENERATOR*
Date: Mon June 10, 1997
Plotname: transient
Flags: real
No. Variables: 9
No. Points: 313
Command: version 3el
Variables:

0	time	time
1	V(2)	voltage
2	V(4)	voltage
3	V(8)	voltage
4	V(6)	voltage
5	V(9)	voltage
6	V(3)	voltage
7	V(5)	voltage
8	V(7)	voltage

Values:

0	0.000000000000000e+00	4	1.600000000000000e−10
	0.000000000000000e+00		4.000000000000000e−01
	5.000000000000000e+00		4.600000000000000e+00
	5.000000000000000e+00		4.990000000000000e+00
	2.000000000000000e+00		2.024000000000000e+00
	0.000000000000000e+00		2.000000000000000e−02
	4.999999930668249e+00		4.982575619838978e+00
	3.466587492675984e−08		3.617594688470175e−08
	5.000000000000000e+00		4.813025039946250e+00
1	2.000000000000000e−11	5	3.200000000000000e−10
	5.000000000000000e−02		8.000000000000000e−01
	4.950000000000000e+00		4.200000000000000e+00
	4.998750000000000e+00		4.980000000000000e+00
	2.003000000000000e+00		2.048000000000000e+00
	2.500000000000000e−03		4.000000000000000e−02
	4.999747404718907e+00		4.923105546201382e+00
	3.484095371176114e−08		3.799788844290367e−08
	4.975189394199576e+00		4.656759335353982e+00
2	4.000000000000000e−11	6	6.400000000000001e−10
	1.000000000000000e−01		1.600000000000000e+00
	4.900000000000000e+00		3.400000000000000e+00
	4.997500000000000e+00		4.960000000000000e+00
	2.006000000000000e+00		2.096000000000000e+00
	5.000000000000000e−03		8.000000000000000e−02
	4.998979414833821e+00		4.600006937881544e+00
	3.501960306267563e−08		4.333224496590186e−08
	4.950765246887764e+00		4.471512663493337e+00
3	8.000000000000001e−11	7	1.280000000000000e−09
	2.000000000000000e−01		3.200000000000000e+00
	4.800000000000000e+00		1.800000000000000e+00
	4.995000000000000e+00		4.920000000000000e+00
	2.012000000000000e+00		2.192000000000000e+00
	1.000000000000000e−02		1.600000000000000e−01
	4.995831451149907e+00		5.542487909118508e−01
	3.538806372226221e−08		3.309933562888388e−02
	4.903123984450634e+00		4.751471744303420e+00

Key Stages of the Software Interface

The key stages in the software interface development and approach followed in each stage with algorithms and pseudocode are explained in following subsections through a data flow diagram of Fig. 3.

1. Read the input files (SPICE input and SPICE output files) and locate the statements in the SPICE code that give rise to generation of input vectors. The input file format of the SPICE simulation is described below.

```
*Title line: SPICE Input File Format
vdd 1 0 DC 5
vin1 2 0 pulse (v1, v2, td, tr, tf, pw, per)
..............other statements.................
vin2 3 0 pwl (t1, v1, t2, v2, t3, v3,.........)
....other piecewise linear (pwl) statements....

The extracted netlist goes here ..........

.tran time_step end_time start_time
.print output nodes (for SPICE2G.6)
.save output nodes (for SPICE3e.1)
.end
```

   PULSE and piecewise linear (PWL) statements, which are followed by the specifications for the pulses to be generated, are the statements considered in reading the input files.

2. Locate PULSE statements and extract the relevant information which includes the pulse width (pw), period (per), and starting voltage (v1) for the vector and the generation of pulses—for example, PULSE (v1, v2, td, tf, pw, per).

   It is to be noted that the risetime (tr) and falltime (tf) values in the PULSE statement are ignored. This is due to the fact that in the testing of an integrated circuit using TekWAVES and LV500, digital stimulus vectors are given as an input and hence the risetime and falltime for these pulses are taken as zero.

3. Locate PWL statements and extract the relevant information, that is, the time instances and the voltage values at these instances required for the generation of pulses—for example, PWL (t1, v1, t2, v2, . . .).

4. Generate input pulses using the parameters in PULSE statements.

5. Generate pulses using the parameters in the PWL statements.

6. Locate the output nodes in SPICE input file.

   The statement .PRINT in the SPICE2G.6 code has been utilized for the purpose of locating output nodes, the reasons being that the statement prints out the voltage values for the nodes specified, as against the line printer plots printed out by .PLOT statement. In the case of SPICE3e.1, .SAVE statement is used to locate the output pulses, since .SAVE statement gives rise to the generation of output vectors in SPICE3e.1.

7. Generate the simulated output vectors using the data in SPICE output file.

8. Print the input and output in EWAV format.

Table 3. EWAV File

```
version event 0 1 0;
date 11 19 1992;
time 9 40 6;
timescale = 1ns;

signal VIN1 input;
signal VIN2 input;
signal VIN4 input;
signal VIN3 input;
signal VIN5 input;
signal V3 output;
signal V5 output;
signal V7 output;

/*RESET Instruction*/
  0: 01010HLH;
 20: 00010LHH;
 40: 00011LHH;
 60: 10111LHH;
 80: 10101LHH;
/*Count to 1*/
100: 01011LHH;
120: 00011HLH;
140: 00011HLH;
160: 10110HLH;
180: 10110HLH;
/*Count to 2*/
200: 01000HLH;
220: 00000LHH;
240: 00001LHH;
260: 10101LHH;
280: 10111LHH;
/*Count to 3*/
300: 01011LHH;
320: 00011HLH;
340: 00011HLH;
360: 10111HLH;
380: 10111HLH;
/*Count to 4*/
400: 01011HLH;
420: 00011LHH;
440: 00011LHH;
460: 10111LHH;
480: 10111LHH;
```

The output file is shown as EWAV file in Table 3. The parameter, for example the timescale, is printed first into an output file (with an extension .ewv). The default time scale for TekWAVES in EWAV format is picoseconds. The time scale is detected from the time instances specified in the PULSE and piecewise linear (PWL) statements, and it is printed into the EWAV file if the time scale is not in picoseconds.

The input and output nodes are printed along with the specification of the direction of node (input, output, or bi-directional). The time instances and pulses corresponding to these time instances are printed according to EWAV format.

The vectors in EWAV format are given as an input to TekWAVES. All the necessary steps including generating cycle boundaries, extracting time sets, pin number allocation, LV500 resource allocation, rules check, and finally the wire guide processing are followed with the help of TekWAVES software, in order to extract the test program necessary for LV500.

The data file containing test vectors in EWAV format is transferred to LV500 through a network communication (ethernet). The chip to be tested is mounted on LV500. The input vectors are passed to the integrated circuit under test. The output digital pulses obtained from the integrated circuit are compared with the simulated output pulses that are present in the test program. The differences between these two are found and reported.

IMPLEMENTATION FOR VLSI CHIP DESIGN VERIFICATION

The input vectors to test logic devices can be obtained from their corresponding truth tables. The input vectors can also be generated from the behavioral testing of digital circuits (9,10) at a high level of abstraction, which includes fault modeling. This test generation method is basically split into four phases: manifestation phase, sensitization phase, propagation phase, and justification phase. The sensitization and justification sequences constitute the test pattern. However, the present work uses input vectors which could be obtained either from the truth table of a logic device under test or from other test pattern generation techniques. As an example, a two-phase nonoverlapping clock generator circuit shown in Fig. 4 is taken as a test case. The steps followed in the implementation of the interface software developed are as follows:

1. The circuit is designed at gate level and then at transistor level in CMOS technology. Figure 4 shows the transistor level diagram.
2. A layout is drawn for the circuit in CMOS technology (11) in MAGIC, a layout editor. A layout can also be drawn by other VLSI CAD tools.
3. The netlist of the circuit is extracted from the layout using the netlist extractor of the layout editor.
4. The input stimulus vectors, in SPICE format, are added to the EXTRACT. The SPICE input file is thus created and is shown in Table 1 (input listing from SPICE2G.6 Output File).

   For the example shown in Fig. 4, the test vectors were generated from the truth table of a two-phase clock generator and were specified in both the PULSE and PWL formats of SPICE simulator.
5. The SPICE simulation is run on the resulting SPICE code using SPICE2G.6 and SPICE3e.1 using the commands:

```
Spice input_file output_file for SPICE2G.6
and Spice3 - b input_file - r rawfile for SPICE3e.1
```

   The output files of the simulation are shown in Tables 1 and 2 (SPICE2G.6 Output File and SPICE3e.1 Output File), respectively.
6. The SPICE input file and each of the SPICE output files (one at a time) are given now as inputs to the software interface and the results are written into EWAV file, which are shown as EWAV file in Table 3.

The first section of EWAV file generated in Table 3 is the environment section in which the data, time and intended destination are optional, but the version at the beginning and the time scale at the end of the section are required. The

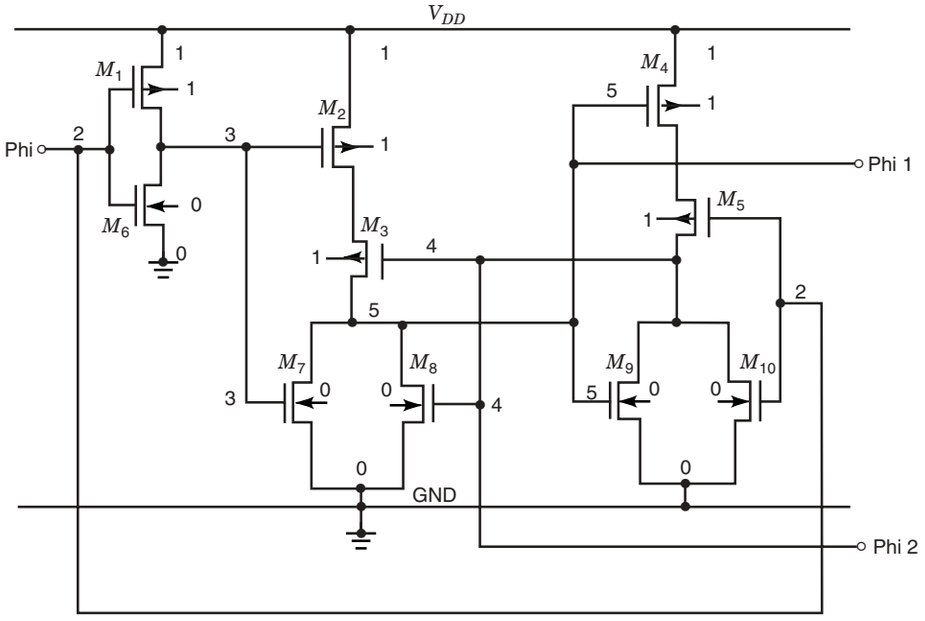

Figure 4. Transistor level diagram of a CMOS two-phase nonoverlapping clock generator.

timescale, if not specified, default to "1 picosecond." The second section shows the list of input and output signals along with the respective polarities. The third section provides the timestamps to the left and data vectors in columns to the right.

ADVANTAGES OF THE INTERFACE

The interface developed between a simulator and design verification system enhances the efficiency of testing by reducing the involvement of human factor. The test engineer now does not need to feed the test vectors by hand to the tester. Furthermore, the test program with clocking and frame information is generated by the interface. Thus, the interface reduces the time consumed in design verification of digital systems.

By virtue of the interface, details such as changing the format of test vectors and including the clocking information in the test program are abstracted away from the user. The user can see the process starting from simulation to testing as a black box, providing the test vectors through simulation in the beginning and running tests on the chip at the end.

CONCLUSIONS

An interface software in "C" language has been developed to integrate SPICE netlist from VLSI circuit layout (along with the test vectors) and TEK L500 ASIC Design Verification System through TekWAVES. The output generated by the software can be given as an input to either TekWAVES or TEK LV500. The formats of both SPICE2G.6 and SPICE3e.1 have been incorporated in CAD tools integration. A graphical user interface has been developed for its efficient use. The utility of the software has been demonstrated through the design of a two-phase nonoverlapping clock generator circuit.

ACKNOWLEDGMENTS

The author acknowledges the IEEE and Gordon and Breach Science Publishers for their permission to use full or part of articles including figures (Ref. 5 published by Gordon and Breach and Ref. 6 published by the IEEE) for the present work.

BIBLIOGRAPHY

1. C. Mead and L. Conway, *Introduction to VLSI Systems,* Reading, MA: Addison-Wesley, 1980.
2. A. E. Ruehli and G. S. Ditlow, Circuit analysis, logic simulation, and design verification for VLSI, *Proc. IEEE,* **71**: 34–48, 1983.
3. J. K. Ousterhout, A switch-level timing verifier for digital MOS VLSI, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **CAD-4**: 336–349, 1985.
4. A. Vladimirescu, A. R. Newton, and D. O. Pederson, *SPICE Version 2G.0 User's Guide,* University of California, Berkeley, September 1980.
5. A. Srivastava and S. R. Palavali, Integration of SPICE with TEK LV500 ASIC design verification system, *J. VLSI Design,* **4**: 69–74, 1996.
6. A. Srivastava and S. R. Palavali, Integration of SPICE with TEK LV511 ASIC design verification system, in *IEEE Proc. 36th Midwest Symp. on Circuits and Systems,* 1993, pp. 673–676.
7. *LV500 Operator's Manual,* Version 1.60, Tektronix, Inc., 1991.
8. *TekWAVES 1.0 User's Guide,* Tektronix, Inc., 1991.
9. E. E. Norrod, An automatic test generation algorithm for hardware description language, in *Proc. 26th ACM / IEEE Design Automation Conference,* 1989, pp. 429–434.
10. J. F. Santucci, A. L. Courbis, and N. Giambiasi, Behavioral testing of digital circuits, *J. Microelectronic Syst. Integration,* **1**: 55–77, 1993.
11. N. H. Weste and K. Eshraghian, *Principles of CMOS VLSI Design,* 2nd ed., Reading, MA: Addison-Wesley, 1993.

ASHOK SRIVASTAVA
Louisiana State University